Algorithms or Actions? A Study in Large-Scale Reinforcement Learning

Anderson Rocha Tavares*, Sivasubramanian Anbalagan*,
Leandro Soriano Marcolino, Luiz Chaimowicz
1 Computer Science Department – Universidade Federal de Minas Gerais
2 School of Computing and Communications – Lancaster University
anderson@dcc.ufmg.br, siva.anbalagan@gmail.com,
l.marcolino@lancaster.ac.uk, chaimo@dcc.ufmg.br

Abstract

Large state and action spaces are very challenging to reinforcement learning. However, in many domains there is a set of algorithms available, which estimate the best action given a state. Hence, agents can either directly learn a performance-maximizing mapping from states to actions, or from states to algorithms. We investigate several aspects of this dilemma, showing sufficient conditions for learning over algorithms to outperform over actions for a finite number of training iterations. We present synthetic experiments to further study such systems. Finally, we propose a function approximation approach, demonstrating the effectiveness of learning over algorithms in real-time strategy games.

1 Introduction

Reinforcement learning aims at developing general agents, which learn by acting directly on the problem action space [Sutton and Barto, 1998]. However, as the state and action spaces grow large, learning agents struggle to attain high performance. On the other hand, many domains have existing algorithms, tailored to the specific problem, and an agent could rely on a pool of algorithms to act on its behalf [Rice, 1976].

Given limited computational resources, however, there is an important conflict: should we learn over actions, training a reinforcement learning agent to discover the best actions to take, or should we learn over algorithms, trying to discover the best algorithm to estimate the best action in each state?

Previous work on reinforcement learning with abstract actions [Sutton et al., 1999; Dietterich, 2000] have shown that the optimal policy may not be attainable when learning over algorithms, although it may accelerate the reinforcement learning process. However, it is still unclear when each method should be preferred. Additionally, having a pool of algorithms may still not enable one to directly apply reinforcement learning techniques when the state space is also very large. In particular, Real-Time Strategy Games are a major challenge for Artificial Intelligence research, given their enormous action and state spaces [Ontañón et al., 2013].

* A. R. Tavares and S. Anbalagan are both first authors.

In this work we establish the conditions where learning over algorithms is helpful, evaluating the sufficient strength of available algorithms, the relation among algorithms and actions set sizes, and possible underlying algorithm creation processes. Synthetic experiments further develop our conclusions. Additionally, we introduce a function approximation approach for Real-Time Strategy Games, demonstrating the effectiveness of learning over algorithms in a complex domain.

2 Related Work

In algorithm selection, one finds a performance-maximizing mapping from problem instances to algorithms [Rice, 1976]. It has been applied to a variety of problems, including SAT [Xu et al., 2008], sorting [Lagoudakis and Littman, 2000], and general video game playing [Bontrager et al., 2016].

Learning over algorithms is also related to action abstractions in reinforcement learning (RL). In MAXQ [Dietterich, 2000], a MDP “calls” other sub-MDPs organized in a graph. Options on MDPs [Sutton et al., 1999] are temporally-extended actions. Algorithms can be seen as one-step options: they can initiate in any state, act according to their internal policy and terminate after one transition. While Sutton et al. [1999] showed that the optimal policy is not attainable when the RL agent has no access to actions, they did not present a detailed study on the dilemma between learning over actions or over algorithms/options. Our theory provides new insights in terms of the required strength of algorithms, relations among algorithms and actions set sizes, and underlying algorithm creation processes.

The use of algorithms, or scripts, is widely adopted in real-time strategy games (RTS) research. For example, in script selection via Monte Carlo planning [Sailer et al., 2007] or assignment of scripts directly to units [Churchill and Buro, 2013] or unit types [Lelis, 2017] via hill-climbing. However, these works are combat-oriented, and do not tackle RTS games as a whole. Recent planning-based approaches, on the other hand, address the full games: AHTN [Ontañón and Buro, 2015] combines hierarchical-task network (HTN) planning, with a minimax-like tree-search algorithm. PuppetSearch combines scripted behavior with game-tree search, by letting the scripts expose a restricted set of actions for the search algorithms to investigate. Two variations exist: PuppetAB [Barriga et al.,
When exploiting, the agent selects the action that maximizes the expected sum of discounted rewards. We consider two RL agents, \( P_1 \) and \( P_2 \), which reason over actions and algorithms, respectively.

### Simple Example

Consider a single state, four actions \( \{a_1, a_2, a_3, a_4\} \) and two available algorithms \( \{x_1, x_2\} \). We assume \( a_1 \) is the optimal action, but that is not known in advance, and the algorithms select each action according to the probabilities shown in Table 1, which results from their reasoning procedures.

<table>
<thead>
<tr>
<th>Algorithm / Action</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0.2</td>
<td>0</td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Given a state, there is a certain probability that the algorithm outputs the true best action, and a certain probability for other actions. Let \( p^a_x \) be the probability of \( x \) selecting an action \( a \). Although we use the pdfs in our analysis, in general they may be unknown. Our analysis allows a deeper understanding of the conflict between learning over algorithms or actions, but a designer may still need to estimate the pdfs when taking a decision between both approaches. There are examples of estimating algorithms’ pdfs in the literature [Marcolino et al., 2013; 2014; Jiang et al., 2014]. Our theoretical analysis is done by comparing the likelihood of selecting the (unknown) optimal action \( a^* \), which maximizes the expected sum of discounted rewards. We consider two RL agents, \( P_1 \) and \( P_2 \), which reason over actions and algorithms, respectively.

### 3.1 Theoretical Analysis

Our main result is a sufficient condition for learning over algorithms to outperform learning over actions. That allows a formal guarantee when learning over algorithms, besides guiding in the number of algorithms used, as we discuss next.

\( P_1 \) or \( P_2 \) selects the best choice (best action \( a^* \) or best algorithm \( x^* \)) with a certain probability \( p_l \). Let \( l \) be the current training iteration. As usual, we consider RL agents where \( l \rightarrow \infty \Rightarrow p_l \rightarrow 1 \). We model \( p_l \) by a learning curve given by the function \( 1 - (\xi_l + e^{\lambda_l} \beta_l)^{-1} \), where \( \xi_l \) and \( \beta_l \) are parameters defining the initial error and the convergence “speed”, respectively. A training process is noisy by nature, but these functions model the average behavior over many training iterations.
events. They converge to 1 in a diminishing returns fashion, as it would be expected in a training process. We then have:

**Theorem 1.** Let $P_i^l$ be the probability that $P_i$ picks $a^*$ at iteration $l$. $P_i^l > P_i^1$ for a finite number of iterations, if $\exists x \in X$ where $p_{a^*}^x > \frac{|X|}{|A|}$. If $l \to \infty$, however, $P_i^l \geq P_i^1$.

**Proof.** If $P_2$ selects $x^*$, it indirectly selects $a^*$ with probability $p_{a^*}^x$. Hence, if $p_{a^*}^x > \frac{|X|}{|A|}$, then $P_2$ selects $a^*$ with higher probability than $P_1$ in the first iteration ($l = 0$). Eventually, however, $P_1$ outperforms $P_2$, since $\lim_{l \to \infty} 1 - \left(\frac{1}{|X|} \times p_{a^*}^x\right)^l = 1$, and $\lim_{l \to \infty} 1 - \left(\frac{1}{|X|} \times p_{a^*}^x\right)^l = 1$. As in the first iteration $P_1$ selects randomly, $1 - \left(\frac{1}{|X|} \times p_{a^*}^x\right)^l = 1$. Similarly, $\lim_{l \to \infty} 1 - \left(\frac{1}{|X|} \times p_{a^*}^x\right)^l = 1$. Therefore, $\xi_1 = \frac{1}{|X|} - 1$, and $\xi_2 = \frac{1}{|X|} - 1$. Hence, if $p_{a^*}^x > \frac{1}{|X|} - 1$, then $P_2$ outperforms $P_1$ until a certain iteration $\tau$. We only need one $x$ such that $p_{a^*}^x > \frac{|X|}{|A|}$, since $p_{a^*}^x \geq p_{a^*}^x$. \hfill $\Box$

We show examples of $P_1$ and $P_2$'s theoretical learning curves in Figure 1. $\tau$ is the training iteration where learning over actions starts to outperform learning over algorithms. Note that Theorem 1 gives us sufficient, but not necessary conditions. That is, if $p_{a^*}^x > \frac{|X|}{|A|}$, we have a formal guarantee that learning over algorithms is better than over actions until a certain training iteration $\tau$. However, there could be cases where $p_{a^*}^x \leq \frac{|X|}{|A|}$, $\forall x$, and $P_2$ still outperforms $P_1$.

For instance, consider 2 actions and 10 algorithms, where $p_{a^*}^x = 0.99, \forall x$. In $l = 0$, $P_1$ picks $a^*$ with probability 0.5, while $P_2$ with probability 0.99, even though $0.99 < \frac{|X|}{|A|} = 5$.

$P_2$ outperforms $P_1$ up to a certain training iteration $\tau$. The previous theorem shows that $\tau > 0$ if $\exists x_p, p_{a^*}^x > \frac{|X|}{|A|}$. We can obtain a lower bound $\tau'$ for $\tau$, by solving the following equation: $1 - \left(\xi_1 + e^{\tau \times \beta_1}\right)^{-1} = \left(1 - \left(\xi_2 + e^{\tau \times \beta_2}\right)^{-1}\right) \times p_{a^*}^x$, since the probability of $P_2$ selecting $a^*$ is at least $1 - \left(\xi_2 + e^{\tau \times \beta_2}\right)^{-1} \times p_{a^*}^x$. Hence, up to training iteration $\tau'$, we have a formal guarantee that $P_2$ is better than $P_1$, if the theorem condition is satisfied. If $x$ is unknown, we derive a less tight lower bound $\tau'' < \tau'$ by solving: $1 - \left(\xi_1 + e^{\tau'' \times \beta_1}\right)^{-1} = \left(1 - \left(\xi_2 + e^{\tau'' \times \beta_2}\right)^{-1}\right) \times p_{a^*}^x$, where $x$ is any algorithm.

Hence, if one is able to estimate $p_{a^*}^x$ (for at least one $x$) and $\beta$, then one would also have a lower bound for $\tau$, leading to a formal guarantee for learning over algorithms up to that training iteration ($\xi$ can be calculated given $|A|$ and $|X|$).

Additionally, note that Theorem 1 does not say that we must have one algorithm whose probability of playing $a^*$ is higher than the probability of playing any other $a$. The condition $p_{a^*}^x > \frac{|X|}{|A|}$ can be valid, even if $\exists a \neq a^*$ such that $p_{a^*} > p_{a^*}^x$. In fact, we can show that learning over algorithms outperforms over actions in a very large action space, even if the probability of an algorithm selecting the best action is very small:

**Corollary 1.** As $|A| \to \infty$, $P_2$ is better than $P_1$ for a finite number of iterations, if and only if $\exists x$, where $p_{a^*}^x > 0$.

**Proof.** Follows from $\lim_{|A| \to \infty} \frac{|X|}{|A|} = 0$, thus $p_{a^*}^x > 0$ satisfies Theorem 1. The “only if” side is trivially true.

Hence, if the number of actions is very large, we only need $p_{a^*}^x > 0$ for at least one algorithm. This result is very relevant even in domains where a designer cannot easily estimate $p_{a^*}^x$.

Interestingly, however, Theorem 1 seems to suggest that the higher the number of algorithms, the worse, as we have that:

**Corollary 2.** If $|X| = 1$, $x$ only needs to play better than uniformly random. As $|X|$ grows, however, the sufficient condition eventually is never satisfied, independent of $p_{a^*}^x$.

**Proof.** Follows immediately from $\frac{|X|}{|A|} \to 1$ for $|X| \to 1$, hence we need $p_{a^*}^x > \frac{|X|}{|A|}$. Likewise, $\frac{|X|}{|A|} > 1$ for $|X| > |A|$, hence we would need $p_{a^*}^x > 1$, which is impossible.

In fact, if there is a fixed algorithm $x$, where $p_{a^*}^x \geq p_{a^*}^x, \forall x \neq x$ in all states, then we should always pick $x$. Intuitively, however, it should be beneficial to have multiple algorithms to choose from. Informally, this may happen because different algorithms may perform better at different states, as discussed in Marcolino et al. [2013]. That is, in many domains we do not have a fixed algorithm $x$ that has a higher probability of selecting $a^*$ than the other algorithms in all states. Therefore, $|X|$ may implicitly also affect the probability of $P_2$ selecting $a^*$, since $P_i^2 \geq \left(1 - \left(\xi_2 + e^{\tau \times \beta_2}\right)^{-1}\right) \times p_{a^*}^x$ (remember that $x^*$ is the algorithm $x$ with the highest $p_{a^*}^x$, across all $x \in X$). Hence, informally, as the size of $X$ grows, we may have a greater chance of adding a new $x$, that has a higher probability of playing $a^*$ than the other algorithms (i.e., $p_{a^*}^x > p_{a^*}^x, \forall 0 \leq i < n$). Therefore, although adding a new algorithm may sacrifice initial performance, it may lead to a higher convergence point (i.e., a higher value for $P_i^2 \geq \left(1 - \left(\xi_2 + e^{\tau \times \beta_2}\right)^{-1}\right) \times p_{a^*}^x$, as $l \to \infty$). A larger value for $P_i^2$ as $l \to \infty$ also increases the number of training iterations where the curve $P_i^2$ is above $P_i^1$. That is, we may have that $\tau_X^2 > \tau_X$, if $|X| > |A|$. Hence, a larger $|X|$ should increase the number of training iterations where $P_2$ still outperforms $P_1$.

Formally, however, it is not true that $\tau$ increases with $|X|$ in general. The way $P_i^2$ changes as new algorithms are added depends on the algorithms’ pdfs. For example, if every new algorithm selects the worst action with probability 1, then $P_2$ just gets worse. If, however, we assume distributions over

![Figure 1: Theoretical learning curves.](image-url)
If the underlying algorithm creation process originates algorithms \( x_i \) with \( p^*_n \sim U(0, u) \), then: (i) \( p^*_n \) grows with \( |X| \) in expectation; (ii) \( \exists x \) where \( p^*_n > \frac{|X|}{|A|} \) in expectation, by following in order of priority: (a) \( |X| \geq 741, |A| > \frac{|X|}{\mu + 3\sigma} \); (b) \( |X| \geq 44, |A| > \frac{|X|}{\mu + 3\sigma} \).

Proof. From the “68–95–99.7” rule, we have: \( p(n^*_p) \geq \mu + \sigma \). Since \( p(n^*_p) \geq \mu + 2\sigma \), we need at least \( t_\sigma \) samples, where \( t_\sigma \approx 1.6448 \). Likewise, for \( p(n^*_p) \geq \mu + 3\sigma \), we need at least \( t_\sigma \approx 44; \) and for \( p(n^*_p) \geq \mu + 3\sigma \), at least \( t_\sigma \approx 741. \mu + 3\sigma \geq \mu + 2\sigma \geq \mu + \sigma \) (the equality comes from \( p(n^*_p) \), being equivalent to \( p(n^*_p) = 1 \)). Hence, \( p(n^*_p) \) grows with \( |X| \), in expectation. Now consider the sufficient condition \( p(n^*_p) > \frac{|X|}{|A|} \). For \( p(n^*_p) \geq \mu + 3\sigma \) in expectation, we need \( \mu + 3\sigma \geq \frac{|X|}{|A|} \). Likewise, for \( p(n^*_p) \geq \mu + 2\sigma \), we need \( |A| > \frac{|X|}{\mu + 3\sigma} \); and for \( p(n^*_p) \geq \mu + \sigma \), \( |A| > \frac{|X|}{\mu + \sigma} \). \( \square \)

Hence, Proposition 2 allows a designer to estimate how many algorithms to use, even without an estimation of \( p_0 \). However, the proposition requires an estimation of \( \mu \) and \( \sigma \), which might come from previous knowledge designing and/or analyzing algorithms for the specific domain.

Fundamentally, however, even if all distribution parameters are unknown, Proposition 1 and Proposition 2 show that under distribution assumptions, one can expect \( p(n^*_p) \) to grow with \( |X| \). Since \( P_\sigma \) converges to \((1 - e^{-e^2} - e^{-4})^{-1} \times p(n^*_p)\), then \( \tau \) also grows with \( |X| \). We study this further in Section 5.

Next, we do not assume an underlying distribution. Instead, algorithms must be chosen from an existing pool \( X (X \subseteq \bar{X}) \).

Proposition 3. Let \( P_{x_i} \) be the probability that \( x_i \) has the highest \( p(n^*_p) \), (across all \( x \in X \)) in a state \( s \). Let \( p_e \) be the expected value of \( p(n^*_p) \) in \( X \). Then, in expectation: (i) \( p(n^*_p) \) grows with \( |X| \); (ii) \( \exists x \) where \( p(n^*_p) > \frac{|X|}{|A|} \), if \( |X| < p_e \times |A| \).

Proof. Given \( n \) algorithms, the probability that at least one of them has the highest \( p(n^*_p) \) (across all \( x \in X \)) is \( p = 1 - \prod_{x \in X} (1 - P_{x_i}) \). Clearly, \( p \to 1 \) as \( n \to \infty \), and thus \( p(n^*_p) \) grows with \( |X| \). However, to satisfy the sufficient condition, we must have \( p_e > \frac{|X|}{|A|} \). \( \square \)

Proposition 3 allows an estimation of the best \( |X| \) as \( |p_e \times |A|| \), given \( p_e \). With a fixed \( X \), in some domains one could estimate \( p_e \) by experimentation over a set of states with a known ground truth. Fundamentally, however, it again shows that one should expect \( p(n^*_p) \) (and \( \tau \), consequently) to grow with \( |X| \). We study this further in the next section.

3.2 Synthetic Experiments

We ran several synthetic experiments, to better investigate the dilemma between learning over actions or algorithms. Each experiment consists of many simulations, where we randomly generate a single state MDP. That is, we sample the mean reward \( r_i \) from \( N(0, 1) \), for each action \( a_i \). When playing \( a_i \), the reward returned is sampled from \( N(r_i, 0.25) \). For each
simulation, we create an agent that learns over actions, and another that learns over algorithms. For both we considered $\alpha$ and $\epsilon$ starting as 1, and decaying at the rate 0.999. We sample $p_a^*\top$ from different distributions, to simulate the creation of algorithms in a given domain.

For each experiment, we run 10000 simulations of 10000 training iterations each. As an example, Figure 2 shows the probability of playing the best action ($p_a^*$) when learning over actions or algorithms, for a Gaussian model (black lines show mean $p_a^*$, and colored areas indicate the standard deviation). Note how the curves follow a similar shape as the ones predicted by our theory (Figure 1). In the appendix, we show that the reward curves also follow a similar shape.

Our theoretical analysis does not yet give the exact number of iterations $\tau$ where learning over algorithms is better. Hence, in Figure 3 we study how $\tau$ changes as several parameters change (problem size $|A|$, algorithm set size $|X|$, $u$ or $\mu$), under a uniform or Gaussian model. A curve beyond the y-axis means that $\tau > 10000$. We repeat the whole procedure 5 times, and the error bars show the confidence interval ($p = 0.01$). When changing one parameter we fix the others (100 actions, 25 algorithms, $u = 0.5$, $\mu = 0.4$). We see that $\tau$ grows with statistical significance, under all parameters considered, for both models. When increasing $u$ and $\mu$ we increase the overall expected performance of the algorithms, and hence this result is expected. It is interesting to note, however, that the curves tend to grow in an exponential fashion.

Concerning $|A|$, the meeting point $\tau$ also tends to grow exponentially. Hence, it gets more advantageous to learn over algorithms as problems grow in complexity. This happens since it gets harder for a RL agent to find the best action, as it requires more exploration. On the other hand, we can still see $\tau$ increasing with $|X|$. That is, even though it gets harder to find the best algorithm, $p_a^*\top$ tends to increase with $|X|$, compensating for the harder exploration, as we discussed in our analysis. Based on Theorem 1, one may expect $\tau$ to drop when $|X| > |A|$. Interestingly, however, we see that $\tau$ tends to converge as $|X|$ grows for both models, instead of dropping (remember that the theorem only gives sufficient conditions).

Our theory focused on $p_a^*$, but the actual reward obtained may be more significant. We evaluated the reward and cumulative reward, and found similar results (shown in the appendix). In the uniform model reward curves, however, we notice that $\tau$ starts to drop when $|X| \gg |A|$.

\[\text{Figure 2: Example of } p_a^* \text{ curves.}\]

\[\text{Figure 3: } \tau \text{ as number of actions, algorithms, } u \text{ and } \mu \text{ grows.}\]

\[\text{Figure 4: Screenshot of } \mu\text{RTS.}\]

4 Learning over Algorithms in RTS Games

We evaluate learning over algorithms in a real-time strategy (RTS) game, which is a very complex domain, where directly learning over actions is impractical. The number of actions for a given state is estimated as over $10^{50}$ [Ontaño et al., 2013]. Hence, we would need over $10^{50}$ training iterations just to explore a single time all the possible actions for a single state. The objective of this section, therefore, is to demonstrate that we can obtain a good performance when learning over algorithms in a complex and relevant domain.

RTS games are adversarial, normally involving resource management, construction, and combat between a large number of military units. They impose a great challenge for AI algorithms, since they have huge action and state spaces and require fast decisions. In this paper we use $\mu$RTS, a simplified RTS game developed for AI research.

A screenshot of $\mu$RTS is shown in Figure 4. In $\mu$RTS, entities are either buildings, units or resources. Buildings are either bases, which can produce workers or barracks, which

---

1http://www.lancaster.ac.uk/staff/sorianom/ijcai18-ap.pdf

2https://github.com/santiontanon/microrts
produce military units. Units are either workers, which harvest resources and have limited combat ability; or military units. Military units are: heavy and light, which are strong but slow or weak but fast melee units, respectively; or ranged, which are long range attack units.

A set of four simple rush algorithms is available in μRTS: (i) Worker: create worker units, have one of them gather resources, and send all others to attack; (ii) Ranged: use a worker to gather resources. With enough resources, build a barrack and generate ranged units, sending them to attack. (iii) Heavy: same as Ranged, but creates the heavy unit instead; (iv) Light: same as before, but creates the light unit. In addition, we implemented two algorithms: (v) BuildBarracks: build a new barrack, allowing faster production of military units; and (vi) Expand: build a new base, increasing the production of worker units and faster gathering of multiple resources. All these compose our set \( X \). In order to handle the large state space, we propose next a Function Approximation approach.

### 4.1 Function Approximation (FA)

In this section, we say that we are taking an “action” \( a \in A \) in a state \( s \) even though we are selecting an algorithm \( x \in X \). This is to follow the traditional notation in RL literature. The main idea of FA is to learn a functional representation of the action-value function \( Q \). This allows us to generalize \( Q \) for similar state-action pairs. We use SARSA [Rummery and Niranjan, 1994], with linear function approximation. Hence, a state \( s \) is represented by a feature vector \( [k_1(s), \ldots, k_n(s)] \), and \( Q(s, a) \) is approximated by \( Q(s, a, w) = \sum_{i=1}^{n} k_i(s) \cdot w_i \), where \( [w_1, \ldots, w_n] \) is a weight vector for action \( a \). The learning problem is to find the best weights for each action. Each time the agent takes an action \( a \), observes the next state \( s' \), and chooses an action \( a' \) in \( s' \), we update \( w_i \) with: \( \Delta w_i = \alpha (r + \gamma Q(s', a', w) - Q(s, a, w)) \cdot k_i(s) \), where \( \alpha \) is the training step size.

The features for a given state of μRTS are obtained as follows: we split the map into 3 × 3 quadrants of equal size. Within each quadrant, the number of units of each type owned by each player \( p \) is a separate feature. Thus, 9 quadrants, 7 unit types and 2 players lead to 9 × 7 × 2 features. Additionally, the cumulative average health of each player’s units within each quadrant is included, leading to 9 × 2 more features. We also include the resources harvested by each player, the current game time and the independent term, with value 1.

Hence, given \( \rho_p = \{u_1^p, \ldots, u_9^p, \ldots, u_7^p, \ldots, u_9^p \} \), where \( u_i^p \) is the number of units of type \( u_i \) in quadrant \( j \) for player \( p \); and \( \beta_p = \{h_1^p, \ldots, h_9^p, \ldots, h_7^p, \ldots, h_9^p \} \), where \( h_i^p \) is the cumulative average health of units of type \( u_i \) in quadrant \( j \); the feature vector is: \( k = [1, p_1, \rho_2, \beta_1, \beta_2, \omega_1, \omega_2, t] \), where \( \omega_p \) is the amount of resources owned by player \( p \), and \( t \) is the current game time. This linear combination of features is replicated for each state \( x \in X \). Hence, we have \( |X| \) equations with \( |k| \) features, leading to \( |X| \times |k| \) weights to adjust. We select an algorithm \( x \in X \) using exponentially decaying \( \epsilon \)-greedy (decayed after every training game).

### 4.2 Results

We evaluate the performance of learning over algorithms using the proposed FA approach. We compare against the state of the art in μRTS: AHTN, PuppetMCTS, PuppetAB, NaiveMCTS and StrategyTactics. They are described in Section 2 and references therein. StrategyTactics won the 2017 μRTS competition, and NaiveMCTS was in the top 5. We used the map “basesWorkers24×24”, and the best parametrization we found: \( \alpha = 10^{-4}, \gamma = 0.9, \epsilon \) exponentially decaying from 0.2 against PuppetAB, PuppetMCTS and AHTN; and decaying from 0.1 for NaiveMCTS and StrategyTactics, after every game (decay rate \( \approx 0.9984 \)). All games have 3000 cycles at most, declared a draw on timeout. Rewards are -1, 0 or 1 for defeat, draw and victory, respectively.

We perform two evaluations. In the first, named Specific, we trained FA in 500 games against PuppetAB, PuppetMCTS and AHTN; and in 100 games against NaiveMCTS and StrategyTactics. The resulting policy is tested against the same adversary that FA was trained against. In the second, named Nemesis, we: (i) trained FA against PuppetMCTS, fixing the resulting policy; and (ii) trained a new instance of FA against the resulting policy of (i), in 500 games. The single resulting policy of (ii) is tested against all adversaries (showing robustness). All tests have 100 games, with \( \alpha = \epsilon = 0 \). We ran 5 repetitions of all experiments, and the error bars show the 99% confidence interval. We consider statistical significance as \( p \leq 0.01 \). Figure 5 shows the results.

In both cases FA significantly defeats all opponents, with win rates higher than 80%. Nemesis and Specific have similar win rates, but Nemesis is significantly better against StrategyTactics. We believe this happens because Nemesis further elaborates on a policy that was already strong (the resulting policy of FA trained against PuppetMCTS).

Allowing algorithm switches at any state could have a negative effect: it could happen so frequently that algorithms would not be able to follow a course of action. Indeed, the agent may switch “too fast” during exploration, but eventually it learns a strong policy, and tends to pick a certain algorithm repeatedly if this leads to higher performance. On the other hand, the algorithm learns to switch to different algorithms when that is more profitable. Figure 6 confirms both situations with the Specific agents, by showing (a) the average number of times an algorithm is chosen consecutively and (b) the average percentage of selections (error bars indicate standard deviation). Hence, all algorithms tend to be chosen, but at different proportions depending on the adversary.

Finally, Figure 7 compares Nemesis and the individual algorithms (\( x \in X \)), when playing against all opponents. We find that our performance is either significantly better than all individual algorithms, or not statistically different from the best algorithm (which is still relevant, since we may not know in advance which one to use), against each adversary. P-values when comparing against the best algorithm \( x \) are: 0.97,
We also thank Tom McCracken for kindly checking our code.

Although action abstractions have been introduced before, our model for learning over algorithms gives novel guidelines backed by a theoretical analysis. Synthetic experiments demonstrate an increase in relative performance with action abstraction and algorithm set sizes. We also introduce a Function Approximation approach for learning over algorithms in RTS games, significantly outperforming state-of-the-art search-based players. The source code of synthetic and \( \mu \)RTS experiments are available at: https://github.com/andertavares/syntheticmdps and https://github.com/SivaAnbalagan1/micrortsFA, respectively. Additionally, the last set of bars shows that all algorithms are individually defeated by Nemesis.

5 Conclusion

Although action abstractions have been introduced before, our model for learning over algorithms gives novel guidelines backed by a theoretical analysis. Synthetic experiments demonstrate an increase in relative performance with action abstraction and algorithm set sizes. We also introduce a Function Approximation approach for learning over algorithms in RTS games, significantly outperforming state-of-the-art search-based players. The source code of synthetic and \( \mu \)RTS experiments are available at: https://github.com/andertavares/syntheticmdps and https://github.com/SivaAnbalagan1/micrortsFA, respectively.

Acknowledgments

We would like to thank Fapemig, CNPq, CAPES, and the School of Computing and Communications for their support. We also thank Tom McCracken for kindly checking our code.

References


